Data Analyst Nanodegree Project 5
December Cohort 2014
Fang
January 20, 2015

Enron Submission Free-Response Questions

Question 1: Summarize for us the goal of this project and how machine learning is useful in trying to accomplish it. As part of your answer, give some background on the dataset and how it can be used to answer the project question. Were there any outliers in the data when you got it, and how did you handle those? [relevant rubric items: "data exploration", "outlier investigation"]

- The enron project primarily provides the classification of persons of interest (poi) where the challenge lies in the limited and spotty nature of the data. There are only 18 identified poi, and 127 non-poi for a total of 145 data points plus the 'TOTAL' outlier which provided a summation of the other features (146 total data points in the original data). Since 'TOTAL' is not a real person, it was removed from the dataset before the classification began.
- For each of the 145 data points there were 21 features in the original dataset with most of them missing values. The most sparse feature is 'loan_advances' with 142 missing values out of 145.

Question 2: What features did you end up using in your POI identifier, and what selection process did you use to pick them? Did you have to do any scaling? Why or why not? As part of the assignment, you should attempt to engineer your own feature that doesn't come ready-made in the dataset--explain what feature you tried to make, and the rationale behind it. (You do not necessarily have to use it in the final analysis, only engineer and test it.) If you used an algorithm like a decision tree, please also give the feature importances of the features that you use. [relevant rubric items: "create new features", "properly scale features", "intelligently select feature"]

- Final Features: *['poi','exercised_stock_options','shared_receipt_with_poi','to_poi_ratio','expenses']*
- I performed two passes to select my features. The basis of both passes was to exclude the features with more than half of their values missing. With the exclusion of *'other'* and *'email_address'* and *'poi'*, I split the remaining 13 features into two classes, financial and e-mail. I then tested the financial features one at a time and ran them through my list of classifiers and settled on features that incremented the performance metrics. I proceeded to perform the same analysis on e-mail only features. I found that in general financial classifiers performed better than e-mail classifiers.
- In the lessons and after running classification tests, I found that e-mail features *'to_messages', 'from_messages', 'from_poi_to_this_person'* and

‘from_this_person_to_poi’ were not a good features for differentiating between poi and non-poi. I created 3 new e-mail class features: *‘to_poi_ratio’*, *‘from_poi_ratio’* and *‘shared_poi_ratio’*. These are the ratio of received or sent messages from poi to the total number of sent or received messages per person. These new features were tested and performed better in differentiating poi/non-poi. In my second pass I primarily focused on the decision tree classifier. I loaded the entirety of the now 16 features, financial and e-mail, and used the gini importance score of each feature provided by the decision tree classifier to reduce the list. As the list of features shrunk, the precision and recall of the decision tree classifier improved.

- No feature scaling was used in the final classification as feature scaling does not affect decision trees. During my first pass using only financial features I was able to add principal component analysis to improve the performance metrics (precision and recall). But the second pass with the addition of the newly created feature and the combination of financial and e-mail features, PCA reduce the performance of the decision tree classifier and was removed.
- The gini importance for the decision tree features are as follows:
  'exercised_stock_options': 0.30
  'expenses': 0.27
  'shared_receipt_with_poi': 0.25
  'to_poi_ratio': 0.18

Question 3: What algorithm did you end up using? What other one(s) did you try? [relevant rubric item: “pick an algorithm”]

- My final classifier was the decision tree classifier. I ran a number of untuned tests with a variety of feature sets against the naive bayes, random forest, adaboost. Naive bayes generally had higher precision but lower recall. Random forest also gave better precision than recall, with precision about twice that of recall. Adaboost provided more even precision and recall scores very close to decision tree scores but the tuning portion of the project made adaboost less desirable as the algorithm tended to take very long to execute. Also setting random state to *None*, created varied scores for the metrics.

Question 4: What does it mean to tune the parameters of an algorithm, and what can happen if you don’t do this well? How did you tune the parameters of your particular algorithm? (Some algorithms don’t have parameters that you need to tune--if this is the case for the one you picked, identify and briefly explain how you would have done it if you used, say, a decision tree classifier). [relevant rubric item: “tune the algorithm”]

- Many classifiers have parameters that can be adjusted to obtain better performance/results. Some algorithms have parameters that allows it to overcome local minimums or run many more iterations for an improved result. A algorithm that is not tuned well could be prone to overfitting or underfitting or even not produce a relevant result. Decision tree classifiers have several parameters that can be adjusted:

max_depth, min_samples_split, max_leaf_nodes, max_features. If the *max_depth* parameter is chosen to be very low, the decision tree would terminate before all leaves will be pure refusing to continue splitting as needed.

- I tuned the decision tree classifier parameter *min_samples_split* after being unable to gain much insight from various decision tree parameters used in grid_search algorithm implemented in sklearn. I iterated over values of *min_samples_split* from 2 to 20 and found a maximum precision and recall around 15 or 16. Overall, there was an increase of about 6 to 11 percentage points on precision and recall of the algorithm. Tuning the *min_samples_split* made intuitive sense to me as we have very low number of samples and I didn't want overfitting due to too fine a split. The optimal parameter of 15 or 16 seems reasonable to me given that there are 18 *pois* and a portion of them are reserved for the test set.

Question 5: What is validation, and what's a classic mistake you can make if you do it wrong? How did you validate your analysis? [relevant rubric item: "validation strategy"]

- Validation is the testing of your model by splitting your available data into non-overlapping training and test sets. This is important because you want to be able to optimize the performance of a classifier with the available data before putting it to the test in the real world. A classic mistake would be to include some of the same data in the test and training sets thus artificially inflating the performance of the classifier. I first used a simple training/test split using the cross_validation package from sklearn. The data was split 60/40 between the training and test sets. The accuracy, precision and recall scores were used to gauge the performance of the classifier.
- Further testing was performed through K-fold cross validation that is provided in the tester.py using the same performance metrics. Precision and recall are also used as metrics of evaluation because accuracy have certain shortcomings. For datasets with skewed classes, in which vastly more examples of one class than the other exist, optimizing accuracy could cause high classifiers to only guess the more represented class in order to raise the performance metric resulting in really poor classification of the lesser represented class. Also in cases where the classifier needs to err on the side of more heavily classifying one class over the other accuracy may not be the best metric of evaluation.
- Precision and recall are measures of relevance. Precision is the accuracy of a classifying a specific element/class given that a particular element is determined: true positives/(true_positives + false positives). Recall is the accuracy of labeling a particular element given that it is that particular element: true_positive/(true_positives+false_negatives)
- The final precision and accuracy for my Decision Tree Classifier with min_samples_split tuned to 16 using tester.py is:
  Precision: 0.48
  Recall: 0.49
  Accuracy: 0.85

References

https://wiki.python.org/moin/UsingPickle

http://scikit-learn.org/stable/modules/pipeline.html

http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

http://scikit-learn.org/stable/auto_examples/ensemble/plot_adaboost_twoclass.html#example-ensemble-plot-adaboost-twoclass-py

https://en.wikipedia.org/wiki/Support_vector_machine

https://en.wikipedia.org/wiki/Naive_Bayes_classifier

http://stackoverflow.com/questions/25475465/how-to-normalize-with-pca-and-scikit-learn

http://scikit-learn.org/stable/modules/generated/sklearn.grid_search.GridSearchCV.html

http://stats.stackexchange.com/questions/69157/why-do-we-need-to-normalize-data-before-analysis

http://scikit-learn.org/stable/auto_examples/calibration/plot_compare_calibration.html#example-calibration-plot-compare-calibration-py

https://en.wikipedia.org/wiki/Hyperparameter_optimization

http://stats.stackexchange.com/questions/122061/relationship-between-gini-importance-and-prediction-performance-say-auc

https://en.wikipedia.org/wiki/Precision_and_recall

https://en.wikipedia.org/wiki/Gini_coefficient

“I hereby confirm that this submission is my work. I have cited above the origins of any parts of the submission that were taken from Websites, books, forums, blog posts, github repositories, etc.”